Digital VLSI System Design

Prof. S. Srinivasan

Department of Electrical Engineering

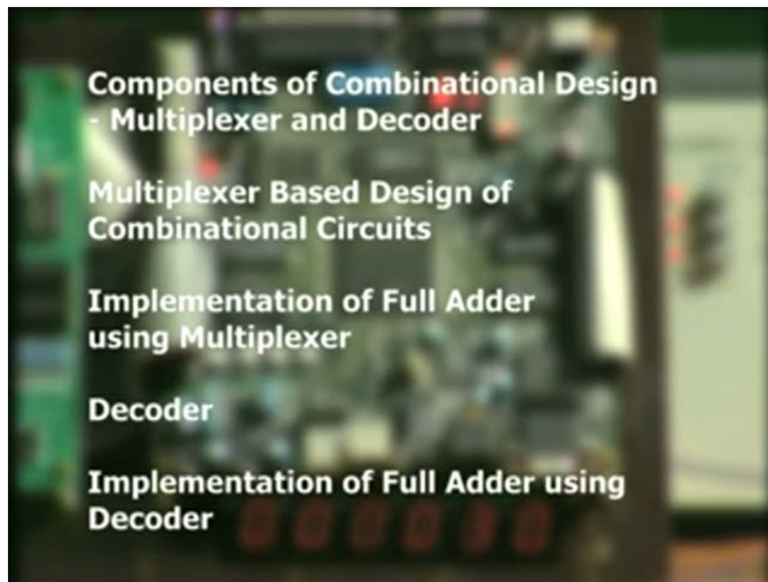
Indian Institute of Technology, Madras

Lecture - 3

Programmable Logic Devices

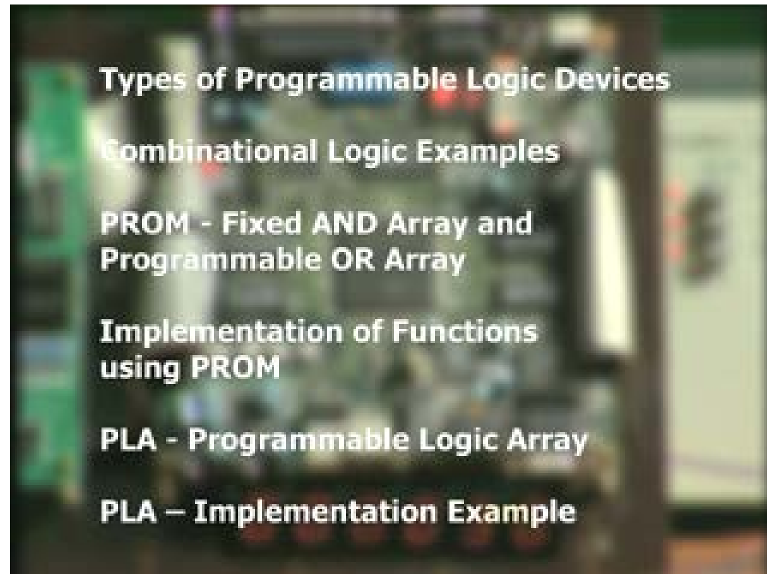
Slide – Summary of contents covered in the previous lecture.

(Refer Slide Time: 01:10)



Slide – Summary of contents covered in this lecture.

(Refer Slide Time: 01:49)



Today, we will talk about programmable logic devices. Programmable logic devices are building blocks which are used to realize digital systems. They can be used to design combination logic; also, they can be used to design sequential logic. Today, we will see some of the basic PLD (Programmable Logic Device) types and take simple combination logic examples. In a later lecture, we will talk about sequential circuit implementation of program using programmable logical devices.

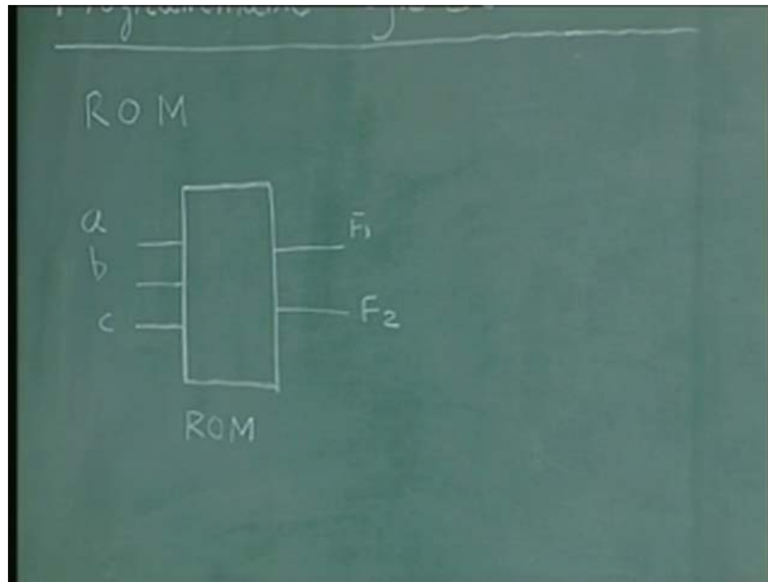
What is programmable logic device? Now we saw in the lecture on multiplexers - combinational logic design using multiplexers - it is easy to make devices or the circuits function-based on the hardware available, rather than go to the karnaugh map minimization technique of reducing the number of gates. Though as circuit complexity increases, we do not want go on building gate level logic; go for higher complex hardware and try to map the given circuit specification to this hardware available.

Now, this circuit becomes more complex than can be accomplished using multiplexers - that is one reason. Another reason is sometimes we want a variation in the multiplexer based design; you want some other type of design. Whatever is the case, either with these

two cases or either increased complex circuit or we want an alternative to multiplexer based design, we have another set of hardware available which are called programming logic devices. These are devices with again gates - AND gates, and OR gates - in large number, available in a single hardware and it is up to you to use them any way you like to realize the function that you want to realize. Another feature of the programmable logic device is that these are, as the names suggest, these are programmable; that means, I can change applications; I can design it for one application, then I can decide to change either the application or specifications of the given application; in either case I can go and re-do the design on this programmable logic device. These programming logic devices are available also as Erasable Programmable Logic devices - EPLDs; when you have an erasable property, you can reuse the hardware either, because, you have made a mistake in the design or because the specification is changed or you want to go for a totally new application. Whatever is the reason, you may able to erase the function of that PLD and then replace it with a new function.

So these are the reasons why we go for the PLD as an alternative to multiplexer based design for combinational logic implementation. There are basically 3 types of program logic devices: one is called the ROM (Refer Slide Time: 05:30) - read only memory – all of us know that; that is you can store the combination of input variables to generate different outputs. In other words, ROM is nothing but a truth table in the hardware form. In a truth table we have the inputs and the combinations and for each combination, the output tells you whether the output is 1 or 0 and they may have several outputs; so ROM is exactly that. ROM will have inputs.

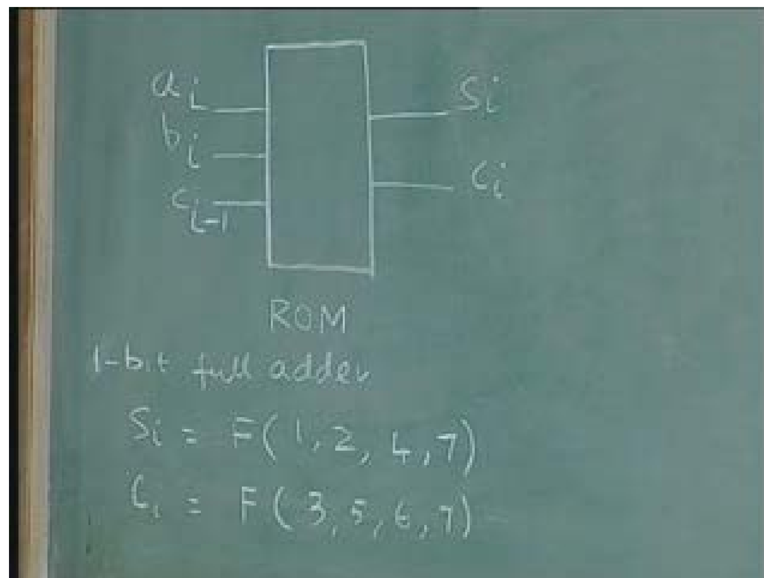
(Refer Slide Time: 06:12)



For example, I may have a, b, c as inputs and I may have 2 outputs called F_1 and F_2. For F_1 and F_2 in the truth table we have rows on which they are 1 and rows on which they are 0.

So you can map all those 0s and 1s for F_1 and F_2 in this hardware and then becomes an implementation. So ROM is nothing but a hardware truth table; hardwired truth table or hardware truth table as you will. The combination for which F_1 is 0 and the combination input for which F_1 is 1, they will be stored in this ROM and whenever the particular combination of a, b, c occurs corresponding output 1 or 0 will come out of the F_1; similarly, for F_2. Now let us take a simple example same - the full adder. I will not write the whole truth table as you are familiar with it.

(Refer Slide Time: 07:34)

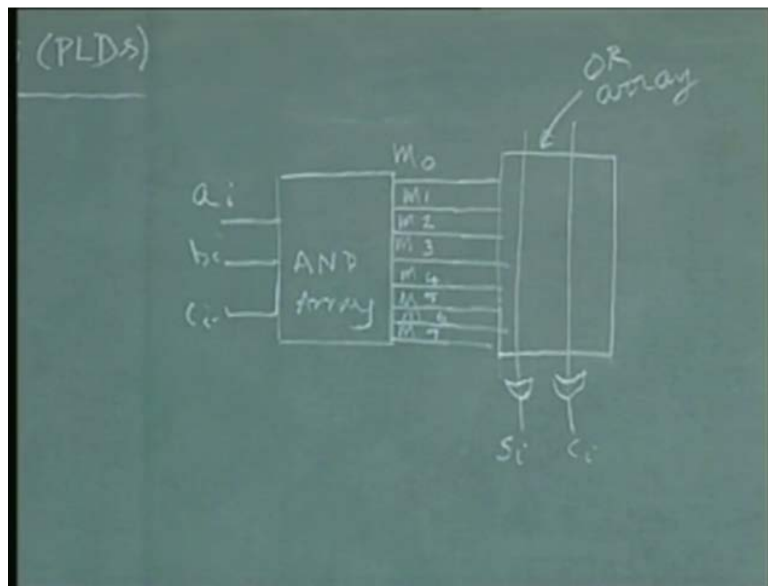


In the full adder circuit - 1-bit full adder - these inputs are a_i, b_i, c_i minus 1; outputs are s_i and c_i (Refer Slide Time: 07:48). We know that s_i is 1; for min terms 1, 2, 4 and 7, sum is 1; for other min terms 0, 3, 5 and 6, sum is 0. Similarly, with carry output, we know that… we will call this F of function is 1 for input 3, 5, 6 and 7 or input combinations 3, 5, 6, 7 - namely min terms 3, 5, 6,7, the carry output is 1; for the rest of the min term combinations the carry output is 0.

Now, this is a truth table implementation directly without making any simplifications. We read this from truth table - I did not write it again because you had it from the previous lecture on multiplexers, the truth tables for sum and carry for inputs a_i, b_i, c_i minus 1 – from that truth table you can read this, because, this is the standard thing. All of you are familiar with this from your first course on digital design, so that is why I do not want to spend more time writing this (Refer Slide Time: 09:40). Now, all I have to do is to take this device, program the device such that whenever the input combination or min terms 1, 2, 4, 7 occurs, the output will be 1 for sum. For the rest of the input combinations or min terms output will be 0 for sum. Likewise, whenever the min term 3, 5, 6, 7 or the input combination 3, 5, 6, 7 occurs it will have the carry as 1. For the rest of the input combination the carry will be 0.

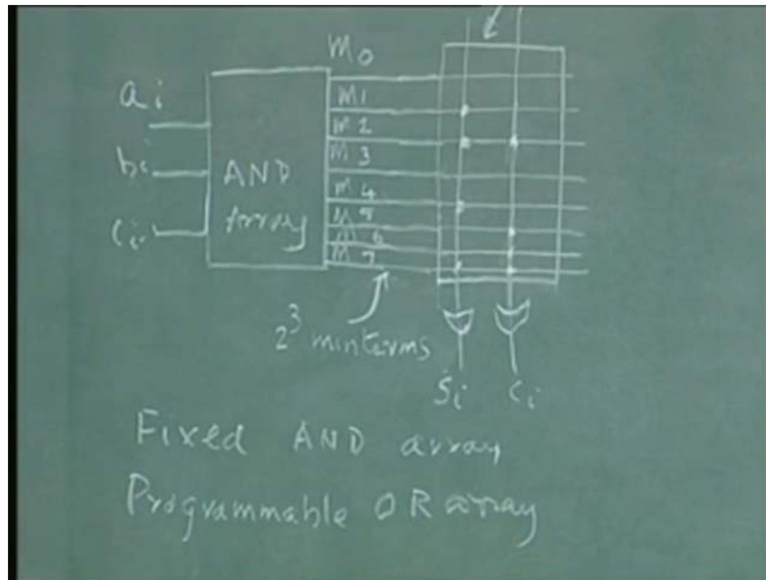
So that is why I am calling it is a hardwired truth table or the truth table in the hardware form. What is inside this ROM? How do you generate these min terms? By combination of a, b and c and how do you generate the output? Output is by combining the min term. So you need a AND gate array, in order to combine this a, b, and c to get different min terms and OR gate array in order to combine the different min terms to form the output.

(Refer Slide Time: 10:55)



Basically it will consist of an AND array in which the inputs will be a_i, b_i, c_i minus 1; output of this AND array will be… corresponding to the min terms I am calling from 0, 1, 2 (Refer Slide Time: 11:30). Actually we should call them… in text book you will see them as m_0, m_1, m_2 it is fine with me… min terms and OR gate array in order to combine this min terms into the required sum and carry (Refer Slide Time: 12:10), usually a symbol of r is put here indicating the min terms are combined in an OR gate. This is only a symbolic representation; I am not showing the actual gate connection inside the AND gate and OR gate. So this is an OR array; this is the AND array. I am not showing the AND gates; it is only a symbolic representation using OR array.

(Refer Slide Time: 12:59)



So as I said now 1, 2, 4, 7 min terms; all the min terms are available, out of which the min terms 1, 2, 4, 7 use the sum and min terms 3, 5, 6, 7. So with this array of AND gates, we are generating the min terms m_0 to m_7 and with these array of OR gates, we are generating s_i and c_i which are functions of the min terms m_0 to m_7 and these min terms are the combinations are a_i, b_i, c_i; a_i, b_i, c_i minus 1 in this case. For example, we know that m_0 is nothing but a_i bar b_i bar c_i minus 1 bar; m_1 is a_i bar b_i bar c_i minus 1; m_7 is for example is a_i, b_i, c_i minus 1. Now, this whole thing (Refer Slide Time: 14:00) can be got in one chip; that is why it is called as programmable. The reason it is called programmable is also I can have the different functions, instead of s_i and c_i, instead of implementing a full adder, I can have some other function in which several other min terms may be involved; I can reconnect my inputs to the OR gate.

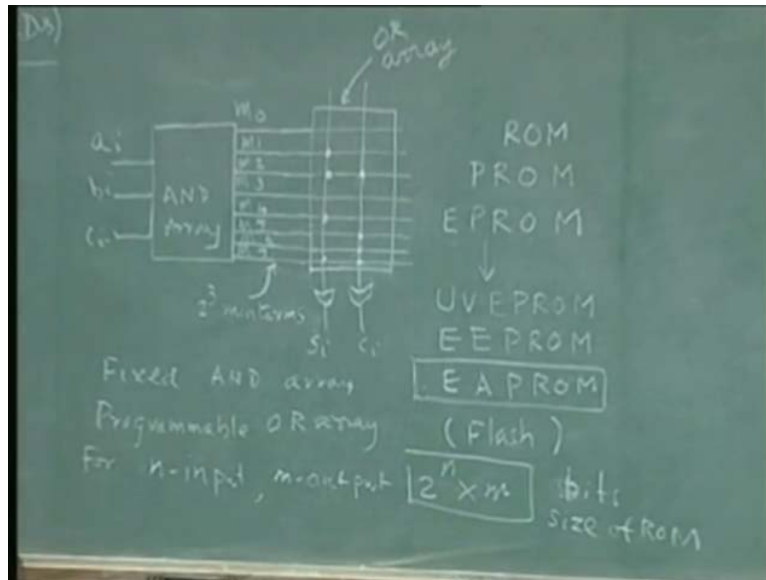
How many min terms would be there depends on the number of inputs and this case there are 3 inputs; so there are 2 power 3 min terms. In general, if there are n inputs you will have 2 power n min terms; number of OR gates will depend on the number of output functions to be implemented. Now we are not showing inside of this; as I is said this is array of AND gates, 2 power n AND gates, in order to generate the each of these min terms and an array of OR gates, in this case, only 2 gates, whose inputs are connected to the various min terms.

Now, why are you calling it programmable? It is a fixed thing. In order to make it programmable I should have a technology by which I can change the connections. These OR gates should have the connectivity which can be changed; if you do not have the connections can be made or unmade, is to be able to connect these OR gates… these min terms are fixed. All the min terms will be generated; moment you have the number of inputs, outputs are fixed, but what way are you going to combine these min terms is the programmability. So this is a program Fixed AND gate array and Programmable OR array.

What is the programmability of this? The programmability comes, because, I can give different inputs to the OR array; that means I should have the technology; that means I should have the gate inputs which can be switched between various min terms. We will not go into the technology of this; that is not in the scope of this design, but I want to tell you in that case you cannot call it ROM - read only memory. When it is a read only memory it is a fixed program, read only memory; once you program it or make the connections to the OR gate, they stay fixed; they stay constant; you cannot change them.

That is the ROM - read only memory, but once you have the programmable feature where I can have the choice of programming the min terms as inputs to the OR gate, such a device would be called a Programmable ROM or PROM.

(Refer Slide Time: 16:44)



That means, I should have some technology by which I can make or break, or connect or disconnect, different min terms as input to the OR gate…. As I said, scope of this lecture will not take us to the technology aspect, but you should know that there is a technology that exists. But I suppose I program it and I do not like the program for some reason - it is because I made a mistake or because I want to change the specification slightly or because I want to totally remove this application and use the same device for some other applications. That means, I should also have the property of erasing the program. Once I have a program, I should able to remove the program and reprogram it for some other applications or some other configurations, so I need also an Erasable Programmable ROM (EPROM). So this is the read only memory where they are Fixed AND gate (Refer Slide Time: 18:03). Even though the OR gates are programmable, the connections are fixed, I cannot change it; that is the ROM; they are the fixed ones; it is all factory programmed, it comes programmed from the factory as it is; use it as it is.

This is the field programmable (Refer Slide Time: 18:17 min); that means, you have the device and the capability to change the input to the OR gates from various min terms; it is called field programmable; that means, I can make the connections the way I like, but once I make it in the field, if I cannot change it, I can make only one set of connections. I need one more feature by which I can remove the program and redo it - this called

erasable programmable; eraser can come out of shining ultra violet rays, is called UVEPROM. You can erase the program after programming it by shining an ultraviolet light on the device. You might have seen EPROMS with the window covered with glass, when you try to open a computer or something microprocessor kit in your lab, you will find a window, this is a window through which we shine the light; that is old technology. Today, we can do it electrically - EEPROM (Electrically Erasable Programmable ROM); by applying electric current pulses, I can erase.

Even these things require the device to be removed from the circuit; you erase the whole thing. I cannot selectively change the program. Suppose I can make it for one application, I can change it for entirely new application. On the other hand, if I have a minor change in my application, I want to selectively change the programming it is not possible in these two techniques of erasure. In the erasure you have to erase the whole thing by ultra violet rays, you have to erase the whole thing by electric pulses.

Today, we have a device called an EAPROM - Electrically Alterable Programmable Read Only Memory - where in, when the device stays in the circuit, without having to remove the device from the circuit, you can selectively alter the programming of the device wherever you want to. These devices are also called Flash memory, you might have heard of this term. These are electrically alterable programmable read only memories or flash memories, where in I can keep the device in the circuit and selectively alter the programming of the device. So that is the most widely used device today.

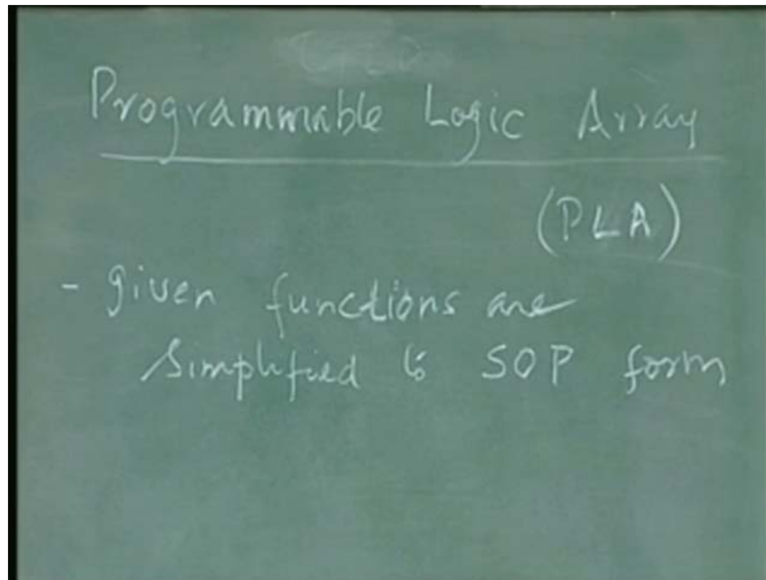
ROM is used in large number of applications because of this size of the ROM required is proportional to the number of inputs to the number of outputs. The number of inputs increases the number of min terms and number of outputs increases the number of OR terms. The AND term explosion is exponential, if I had one more input instead of 3 inputs if there were 4 inputs, I will have 2 power 4 which is 16 min terms. So every one extra input I need a 2 power addition in the number of min terms. On the other hand, OR terms will increase only by the number of outputs. So, in general, for n input and m output circuit, I need 2 power n times m – this is the size (Refer Slide Time: 22:35). Size

is given in bits; this is the size of the ROM for n input and m output device, the size will be 2 power n times m.

Remember that the input increases the size exponentially, whereas the output increases only linearly. This is a very simple technology and today as I said in the beginning, the cost of these gates it is not at all considered. We have semiconductor complexity running into millions of gates, billions of transistors and a device like a 1 bit ROM can be very easily made; so, you do not have to the really optimize on these. Except in very huge applications, where the number of inputs are extremely large and this becomes exponentially large. We do not know want to use that size just like that, because, even though it is available, bigger area is available, it increases the cost of the device, but the cost of the whole circuit or device system you are designing plus you will be unutilized. You have only a large number of min terms, but how many of them are used?

The number of min terms being used by a given function will be usually very small. Any function, whether 3-input function, 4-input function, 5-input function or even 57 input functions, whatever the number of inputs, the given output will have only a few selected terms out of it. That means most of the min terms that are generated will not be used in the output. This is the waste of the hardware or silicon real estate as I call it. So, for these two reasons - that is, under utilization of the size and exponential growth in the size - for very large applications, you want to have other alternatives.

(Refer Slide Time: 24:39)



The second alternative for program logic device family is called Programmable Logic Array or PLA for short. Now, the difference between the ROM and PLA approach is simple. Again it is programmable, in the sense I should be able to change my connectivity either for making minor changes or because I want to correct a design bug or make a minor change in the specification or totally remove the application and put in a new application; so these are also available as erasable programmable. This (PLA) is also erasable available; so they are called EPLDs - Erasable Programmable Logical Device. Any way that is for all devices that is common. So I will not write it separately for this.

Now the question is what is the difference between PROM or a ROM…. We will call it ROM even though it is a PROM; we will use the programmable read only memory, but generally it is called ROM based design; they do not say PROM based design; you do not hear that term. PLA and ROM - what is the difference? I said number of min terms are generated, many of them are not used, especially when the inputs are large. So, can we now generate only such of those terms which will be used or can we only generate a few product terms which will be used in my output OR gates?

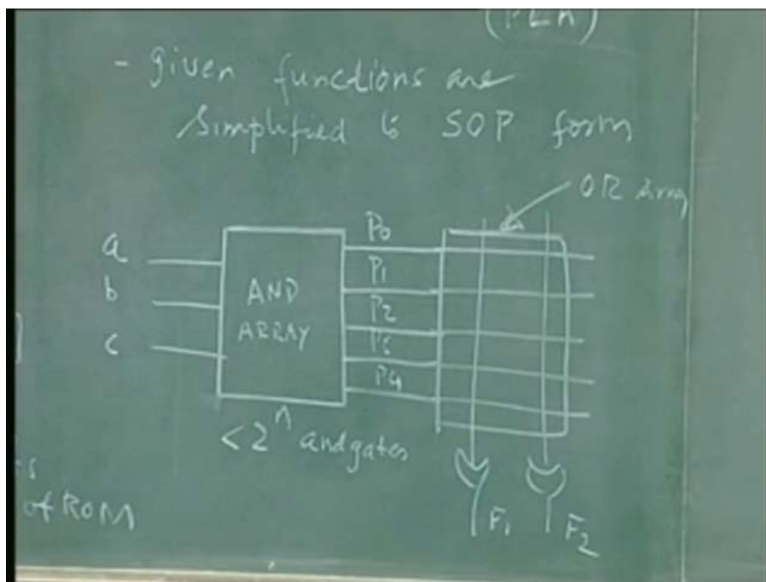
In this case, we did not have any simplification. The truth table directly implemented in the hardware form. So that is why I called it hardwired truth table. But in the program

logic array case, I will use a few select product terms in order to generate the required output; that means, I do not have to have the whole set of AND gates, 2 power n, I will have a fewer number of AND gates and generate a bunch of product terms which will be used in the implementation of the final output.

Now, the question is - how do you select what product terms to be used? There are two things: one is I am going to reduce the number of AND gates, so the size of the device is going to reduce for the given application or when the number of inputs are very large, I do not have to generate all the AND product terms or a min terms, but I will only generate a few product terms.

You will have to decide what product terms to generate. That will require karnaugh maps. I cannot do guesswork of identifying the product terms to be used in the final. So what I do is to the given problem or functions (Refer Slide Time: 28:00), I will use functions as there can be more than one function, are simplified to the certain extent, to sum of product form. Sum of product is where you have to have a few product terms whose sum will be the output. In doing so what we will do is, I am going to generate the few product terms.

(Refer Slide Time: 28:39)

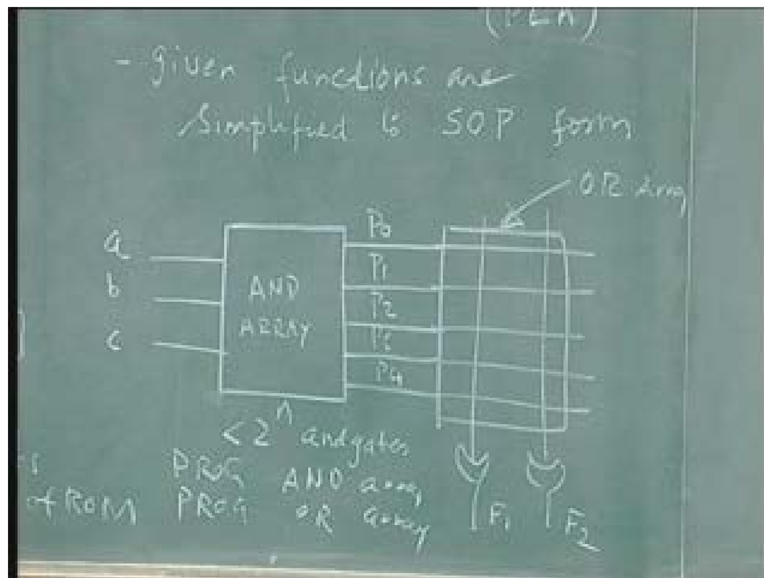


So this will be the configuration of the program logic array. I will have inputs, I will call them a, b, c. Normally, I require 8 product terms as min terms, if it is the ROM approach. Let us say this case I will have only 5 outputs instead of calling them m_0, m_1, m_2, m_3, m_4 and I am going to call them p_0, p_1, p_2, p_3, p_4; p stands for the product terms, m stands for the min term. You know the difference, these things you would have learnt from the first course in the digital design.

Product term need not contain all the inputs or either in the true form or complement form. The min term should contain all the inputs either in its true form or complement form. Product term may contain some or all of these inputs. Now, I am going to have as an example 5 product terms; I am going to have OR combination which will combine by product terms to generate the required functions F_1 and F_2. So this case, I am going to say this is an AND array, but it should not have 2 power n min terms, this will have only less than 2 power n AND gates. How small depends on the output terms and given number of inputs and these things are combined in the same way as we did in the previous case; then OR array; there is no change in this.

So instead of having the fixed AND array, which will generate all the min terms and a programmed or programmable OR array in which I will program the inputs of the OR gates, I am going to have a programmable AND array in which I will program the product terms the way I like and combine them again in a programmable OR array by which I can selectively use the generated product terms in order to realize my outputs.

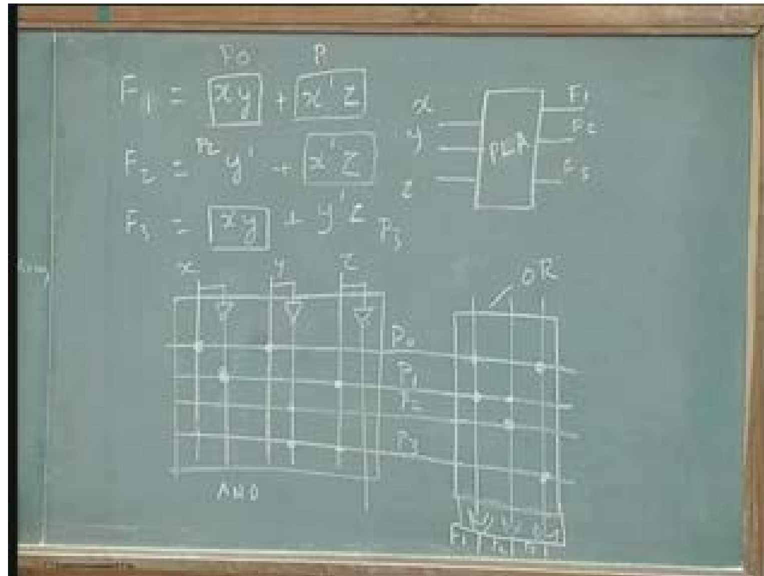
(Refer Slide Time: 31:18)



So this is called programmable AND and programmable OR; programmable AND array and programmable OR array. How do you select p_0, p_1, p_2, p_3, p_4 depends on the functions and cells. I need to do little bit of simplification in order to get them. One other thing I would like to do is, which is interesting is when I select the p_0, p_1, p_2, p_3, p_4, I will select them such that they have as many of these things are common to the two outputs.

For example, if I generate one of the terms p_3 as common to F_1 and F_2 it will be used 2 times. My idea is to use the hardware to the maximum possible extent; that is my idea in my design. So if I can simplify the product term, simplify my input functions or output functions F_1 and F_2 using the inputs a, b, c in such a way that I have as many common terms as possible, then I can make an efficient implementation of F_1 and F_2. You will see in the example in this case I am not going to the karnaugh map, because as I said this is something you must have learnt in the previous course.

(Refer Slide Time: 32:23)



So I am going to take 3 functions F_1, F_2, F_3 we will be calling F_1 is xy plus x bar z; $F_2$is y prime plus x bar z; F_3 would be xy plus y bar z. How do I get this? I have an input of x, y, z and an output of F_1, F_2, F_3 and I want PLA - programmable logic array - to be used for this function (Refer Slide Time: 33:25).

How do I get F_1, F_2, F_3? From the problem specification, either they have been given to me in this form or in some other form like a truth table or a tabular form, from which I got this by simplification using karnaugh map. I am not showing that step because that is too trivial for you in a second course in the digital design.

What is interesting is though the terms which are common - x bar z is common. How many terms would I use if I do not have the common terms? I would have generated 6 product terms p_1, p_2, p_3, p_4, p_5, p_6.

Now, since these 2 are common to, xy is common to F_1 and F_3; I am saving on one product term; x bar z is common to F_1 and F_2, and I am saving to other product term; so I now use only 4 product terms 1, 2, 3, 4. From 6 product terms, I am going to use 4 product terms, saving on 2 product terms.

How did I get it? In this case, I have said that this function was given to me; it could have been simplified or sometimes you may have to be careful in simplifying it yourself; you do not over simplify. Sometimes in a karnaugh map approach, the traditional approach is to simplify to the minimum possible form. I would minimize such that I use the minimum of the AND gates and minimum number of OR gates and minimum number of inverters. In this case, I will not do it. In this case, I will look for common terms even if it is comes to be more terms, as long as there are more and more common terms, I will prefer that option because, in the end is going to give me less hardware. What I am trying to say is if I had reduced it I would have got 5 terms, let us say; but by keeping some terms and not reducing, I am getting 2 pairs of common terms, which means I am going to use only 4 AND gates.

So, 4 AND gates is preferable to 5 AND gates. So that the extra step of simplification from 6 to 5 is not worth it. So this is the technique you have to practice; I am not going to go into the further details. I will have x, y (Refer Slide Time: 35:57 min). This is how it is connected; x, y, z; x bar, y bar; these are all common inside; the PLA will have all this. So my first product term, p_1 will be xy; so this is x, y; p_1 and p_0. Second product term would be x bar z, p_1; I am going to call this p_2; y bar; this will be p_3; y bar z (Refer Slide Time: 37:09). So 4 product terms p_0, p_1, p_2, p_3, feeding into 3 OR gates. First F_1 is p_0 or p_1; this is my F_1. Second term F_2 would be p_2, this is same as p_1 (Refer Slide Time: 38:09), p_1 or p_2; F_3 would be… this is xy same as p_0 and p_3. So now, this term is common to two terms of the output (Refer Slide Time: 38:24). This product term is common to two terms; this is common to two of the outputs and these two are single. So this is F_1, F_2, F_3. This is how you do this F_1, F_2, F_3.

Now the whole device should be available to a single package; you do not know what is inside this. This is the AND array (Refer Slide Time: 38:51 min) along with the invertors to the input and this is the OR array. So as far as you are concerned, 3 inputs x, y, z are giving rise to 3 outputs F_1, F_2, F_3. This is also part of the array, the OR gate, like that and the fact that you have simplified such that the common terms are available between different F_1, F_2, F_3 functions, I can reduce the number of product terms I need to generate. That is the trick in programmable logic array design; programmable array logic design is

simplified such that as few product terms are possible totally, not for individual functions; that is the only thing. That means, number of AND gates is reduced; there are only 5 here; there are 4 here instead of 3 inputs we would have used 8 normally; instead it is only4. There is no reduction in the number of OR gates. Whatever AND gates are more fully used, I am using them better, so the silicon is not wasted.

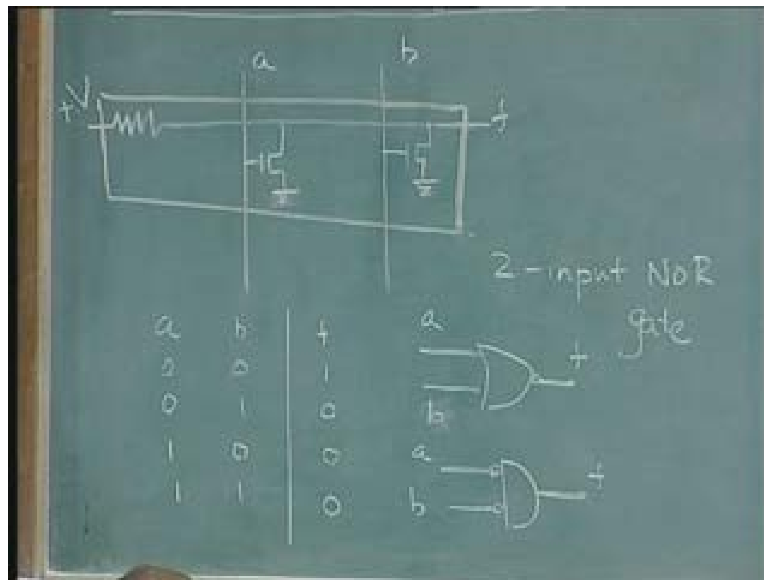
This is the output function (Refer Slide Time: 40:10); this approach is programmable logic array approach, which is slightly better than the ROM approach. In the case of the ROM, PROM, EPROM UVPROM or EAPROM, I told you that it is not in this scope of this lecture series to talk about the technology of these devices; how exactly the making of the contacts is made; how is an erasable program built in ROM; what is the device that makes the programmablility to be erased and what is the difference between EAPROM and EEPROM; all those things I deliberately left out because, it lot of device physics and device theory you have to get into. But in the other hand programmable logic array I have talked about in the last few minutes I want to give one technique. It might look like it is also device level, but it is not; it is really a circuit technique.

So I want talk about little bit of the hardware of the programmable logic array. Not so much because I want to tell you how the devices are made, but more because it has an interesting circuit aspect to it. Since the designer should also completely know about the circuit implementation aspects of a programmable device, we will spend a few minutes on the actual way in which the programmable logic array designs are implemented. Even though I drew an AND gate array and OR gate array in the example, in practice, you do not make AND gates and OR gates within a chip, within a programmable logic device chip or programmable logic array chip. There will not be AND gates and OR gates. They generally use one type of gate either a NOR gate or a NAND gate.

NOR gate and NAND gates are known as universal gates and for reasons of technology they either, prefer NOR gates or NAND gates. Let us say, this device we are talking about - the programmable array logic or programmable logic array device - uses only NOR gates throughout. I should be able to implement my AND gate array, OR gate array

everything using my NOR gates. How it is really done within the circuit? I will just like to explain to you briefly.

(Refer Slide Time: 42:46)



Consider this diagram; this is a line or a bus, you call it, connected to a voltage - positive voltage (Refer Slide Time: 42:54). There are two junctions in which you can make the junction or not, depending on these devices; a and b are the inputs, this is output. Actually, this is how it will look from a user's point of view. So, there are two inputs a and b and output f connected to a positive voltage v and now when both are 0 - a is 0, b is 0, these switches (Refer Slide Time: 43:31) are nothing but MOSFETs, n channel MOSFET. So n channel MOSFET will conduct when the gate voltage is high and this will be cut off when the gate voltage is low. So we will only talk about the cut off and saturation region; we will not talk about the triode region etc.; this is not a device theory class. So this n channel MOSFET can be turned on or tuned off; let us say we are putting a voltage here.

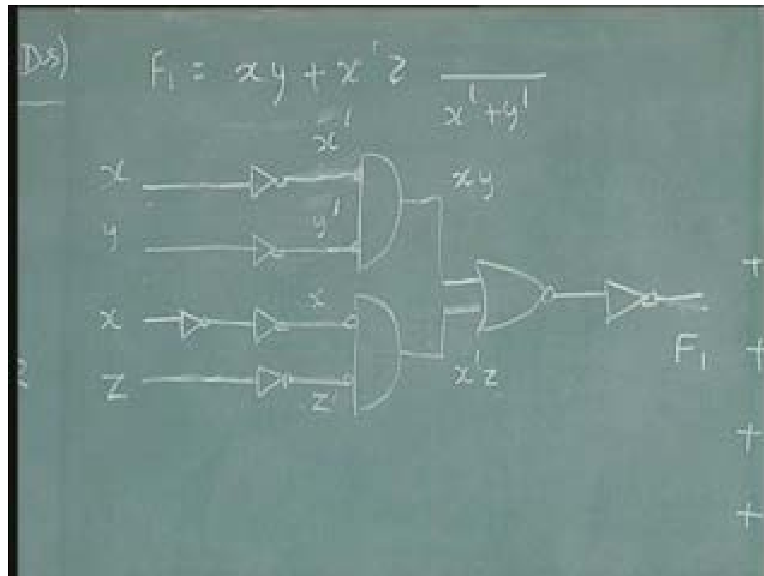
Putting a voltage 0 here (Refer Slide Time: 44:01), you turn off this device; putting a voltage 1 here, will turn this device on. So when both are 0, both the connections are not there; that means, these both are turned off, the entire voltage v appears at the output f. So when input is 0 0, the output is 1, even when one of these devices is on either because a is

high or b is high. Whatever is the case, as soon as one of the devices is on, there will be a ground part because, this is connected to the ground, this bus line is connected to ground, through the device (Refer Slide Time: 44:44).

The device is on device - bus connected to ground, if a is on; if b is on, bus connected to ground through this and both are on of course, is going to be connected to the ground. So if a is 0 or b is 1 or b is 0 and a is 1, either case the output going to be 0 and also when both are 1, output is going to be 0. I can realize an output f as 1 0 0 0 for input combination 0 0 0 1, 1 0 1 1 by this arrangement and this is nothing but the characterization of NOR gate. All of you know that complement of OR gate. OR gate has 0 1 1 1. NOR gate is a complement of OR gate is 1 0 0 0; this is circuit symbol for that (Refer Slide Time: 45:26).

Now this is another symbol (Refer Slide Time: 45:32). These two are symbols, two inputs inverted gets a NOR function in an AND gate. In an OR gate output inverted using NOR function; both are NOR gates. So this is a 2-input NOR gate. I said the programmable logic array device, all the gates inside are NOR gates even though you may have an AND array and an OR array; of course, invertors; inverters you know that NOR can be connected as an inverter, NAND can be connected as an inverter; we know all that from the previous course in digital design. Now, I have to tell you how I can get whatever function I want for f_1, f_2, f_3 what ever, using inputs a, b, c, etc., using only a complete NOR gate combination. How is this done within the device?

(Refer Slide Time: 46:44)

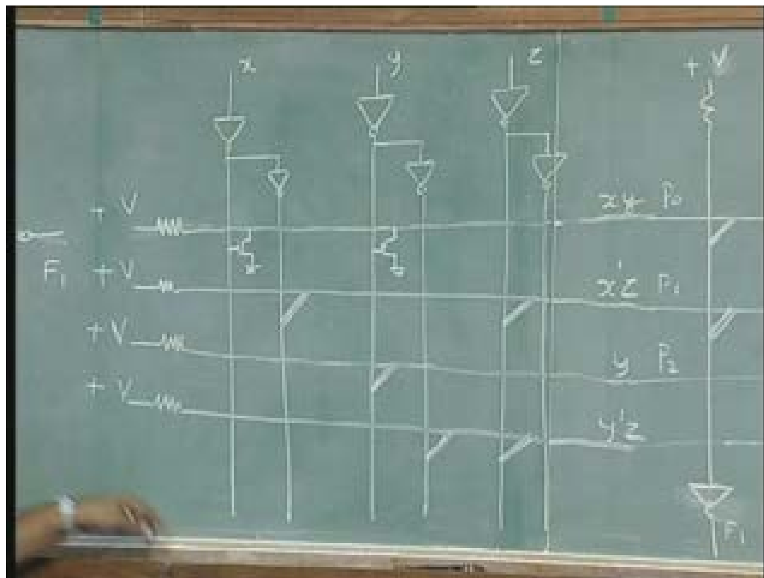


Let us take the example we took; I had three functions - F_1, F_2, F_3 and three inputs x, y, z; I am going to take the first one of those functions: F_1 as x y or x bar z, this is the $F_1$of the previous example. Now since these have to be NOR gates, I have no choice; this has to be a NOR gate. I can only connect a NOR gate to another NOR gate; so I will do something to the input and some of the output and manipulate the whole thing. I do this by connecting invertors. So I have an inverter at each of the inputs, if there is an inversion required I put in an inverter; if the inversion is not required I put in a double inverter. That means, here it is x, I put in inverter x bar y bar, this is x, two inverters of x, z, two invertors of z, z within an inverter z bar. So the output here is x bar or y bar whole bar, same as x y (Refer Slide Time: 47:32). You know that from Demorgan's theorem, again from the first course in digital design. Those of you are watching this series, I assume that you are very clear about the first course and digital design together. I am going to use lot of results, make a lot of assumptions; so if you are not familiar with the basic digital concepts, I recommend strongly that you go through the digital design course completely; take a standard text book or take a standard video series, look at all these chapters; make yourself very thorough, only than you get full benefits out of these lectures.

Now you have x; similarly you have …. I am not going to do everything; this going to x bar z because I have x becomes x here (Refer Slide Time: 48:24); z, z bar; so it gets an inverter. Now this NOR gate with an inverter becomes an OR gate. So an output is x, y or x bar z which is F_1 .

This inversion is taken care of by the inverter. So this is the philosophy of using NOR gates through out, implementing whatever function you want. If you go back to the example we did F_1, F_2, F_3 with three inputs x, y, z; three outputs F_1, F_2, F_3, this is what the circuit is going to be.

(Refer Slide Time: 48:56)



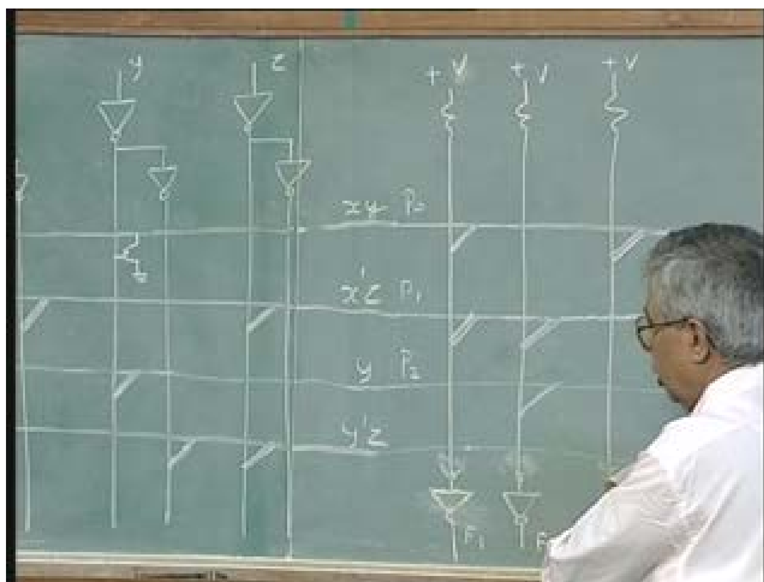
I am going to have at each of the inputs an inverter; this is the original inverter which will be the original configuration of AND OR combination I use x, x bar, y, y bar ,z, z bar with an inverter.

Now putting an extra inverter; that means, this becomes x bar and this becomes x bar bar which is x. This is x, x bar; y, y bar; z, z bar (Refer Slide Time: 49:30) and because of the same logic of the inversion of the inputs because of the NOR gates, these switches implemented in the NOR gate function as just now explained, even though x bar is connected to this and y bar is connected to this, because of an inversion this becomes xy. The same logic I talked about earlier; likewise this x and this z bar connected makes it x

bar z. This is my product term p_0; this is the product term p_1; the third product term p_2 is y bar y; so you do not have to worry about this y bar and y all that. If completely remove this (Refer Slide Time: 50:17 min), you feel very comfortable because one inversion, another inversion, all that, you do not want get confused. You simply say this is xy; so this is x y; this is x bar z circuit x bar z; that inversion takes care of it. Here it is y, so here it will be y bar z. This will be further same type of NOR gate configuration, will give me the required functions, but I need an inverter finally; a jet inverter. The inverter of the OR gate; the NOR gate inverted into an OR gate - that inverter; these inverters correspond to that, so there is an OR gate which I am not showing. Then OR gate and s; so here will be a… (Refer Slide Time: 51:07 min); there is no point showing it here because the NOR gate is only a symbolic representation.

The actual circuitry comes because of the… I do not know want to really show this because it will only confuse you. This was in the original drawing, this is schematic (Refer Slide Time: 51:18), but now this is the real drawing, so because I am going to pull these down (Refer Slide Time: 51:27 min) and everything this becomes NOR with the inverter becomes F_1, F_2, F_3. So this is the actual way it is implemented inside.

(Refer Slide Time: 51:36)



This is not hardware. As a said I am not going to talk too much about the implemented technology aspects of the programmable logic devices, but this is the circuit technique we are using, because I am given a function of AND OR inverter, I am asked to implement using NOR gates; I need to have these type of circuit technique. So the circuit technique of converting the AND or OR inverter to a NOR logic family is what I wanted to explain to you in this technique (Refer Slide Time: 52:05). This is the design using programmable logic array.

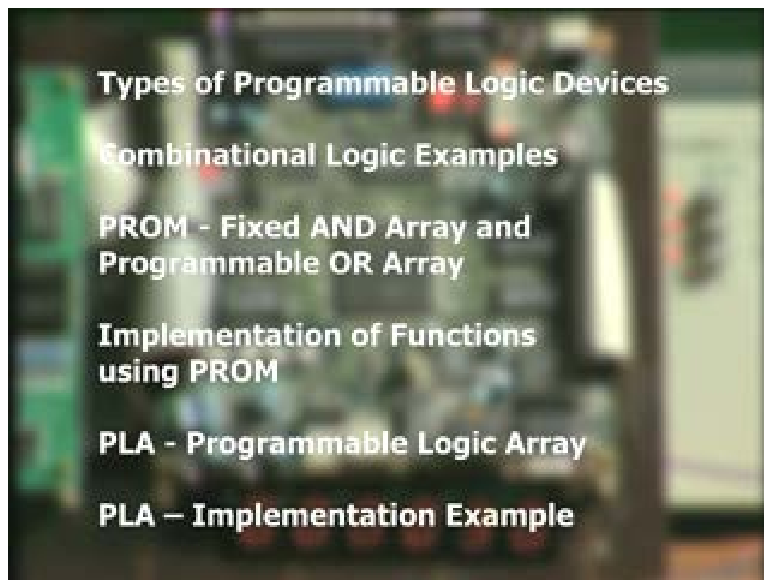
We have now today seen PROMs, read only memories and programmable read only memories; how to implement functions using that; also some programmable logic arrays and how to do program the logic array implementation of digital systems.

Thank you.

Summary of Lecture 3

Programmable Logic Devices

(Refer Slide Time: 52:55)



Next Lecture

Programmable Array Logic

(Refer Slide Time: 53:30)